Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/FR05/000323

International filing date: 11 February 2005 (11.02.2005)

Document type: Certified copy of priority document

Document details: Country/Office: FR
Number: 0401479
Filing date: 13 February 2004 (13.02.2004)

Date of receipt at the International Bureau: 08 April 2005 (08.04.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)

R E P U B L I Q U E F R A N Ç A I S E

PCT/FR/2005/00323

INPI
INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE

BREVET D'INVENTION

CERTIFICAT D'UTILITÉ - CERTIFICAT D'ADDITION

COPIE OFFICIELLE

Le Directeur général de l'Institut national de la propriété industrielle certifie que le document ci-annexé est la copie certifiée conforme d'une demande de titre de propriété industrielle déposée à l'Institut.

Fait à Paris, le 16 FEV. 2005

Pour le Directeur général de l'Institut
national de la propriété industrielle
Le Chef du Département des brevets

Martine PLANCHE

INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE
SIEGE
26 bis, rue de Saint-Petersbourg
75800 PARIS cedex 08
Téléphone : 33 (0)1 53 04 53 04
Télécopie : 33 (0)1 53 04 45 23
www.inpi.fr

ETABLISSEMENT PUBLIC NATIONAL CREE PAR LA LOI N° 51-444 DU 19 AVRIL 1951

1er dépôt

INPI
INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE
26 bis, rue de Saint Pétersbourg
75800 Paris Cedex 08
Téléphone : 33 (1) 53 04 53 04 Télécopie : 33 (1) 42 94 86 54

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

Code de la propriété intellectuelle - Livre VI

cerfa N° 11354*02

REQUÊTE EN DÉLIVRANCE page 1/2

BR1

Cet imprimé est à remplir lisiblement à l'encre noire DB 540 @ W / 010801

Réservé à l'INPI

REMISE DES PIÈCES DATE 13 FEV 2004
LIEU 69 INPI LYON
N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI 0401479
DATE DE DÉPÔT ATTRIBUÉE PAR L'INPI 13 FEV. 2004

Vos références pour ce dossier *(facultatif)* 706020c3SLC/AMD

1 NOM ET ADRESSE DU DEMANDEUR OU DU MANDATAIRE À QUI LA CORRESPONDANCE DOIT ÊTRE ADRESSÉE

Cabinet BEAU de LOMENIE
51, avenue Jean-Jaurès
B. P. 7073

69301 LYON CEDEX 07

Confirmation d'un dépôt par télécopie ☐ N° attribué par l'INPI à la télécopie

2 NATURE DE LA DEMANDE	Cochez l'une des 4 cases suivantes
Demande de brevet	☒
Demande de certificat d'utilité	☐
Demande divisionnaire	☐
Demande de brevet initiale	N° Date
ou demande de certificat d'utilité initiale	N° Date
Transformation d'une demande de brevet européen *Demande de brevet initiale*	☐ N° Date

3 TITRE DE L'INVENTION (200 caractères ou espaces maximum)

Procédé d'élaboration automatique de fichiers de description HDL de système électronique digital intégré et système digital électronique intégré obtenu

4 DÉCLARATION DE PRIORITÉ OU REQUÊTE DU BÉNÉFICE DE LA DATE DE DÉPÔT D'UNE DEMANDE ANTÉRIEURE FRANÇAISE

Pays ou organisation
Date N°
Pays ou organisation
Date N°
Pays ou organisation
Date N°

☐ **S'il y a d'autres priorités, cochez la case et utilisez l'imprimé «Suite»**

5 DEMANDEUR (Cochez l'une des 2 cases)		☒ Personne morale ☐ Personne physique
Nom ou dénomination sociale		INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE
Prénoms		
Forme juridique		
N° SIREN		|_|_|_|_|_|_|_|_|_|
Code APE-NAF		|_|_|_|_|
Domicile ou siège	Rue	Service Recherche et Valorisation 46, avenue Félix Viallet
	Code postal et ville	38031 GRENOBLE CEDEX 1
	Pays	France
Nationalité		Française
N° de téléphone *(facultatif)*		N° de télécopie *(facultatif)*
Adresse électronique *(facultatif)*		

☐ **S'il y a plus d'un demandeur, cochez la case et utilisez l'imprimé «Suite»**

Remplir impérativement la 2ème page

1er dépôt

INPI
INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

REQUÊTE EN DÉLIVRANCE
page 2/2

BR2

Réservé à l'INPI

REMISE DES PIÈCES DATE 13 FEV 2004
LIEU 69 INPI LYON
N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI 0401479

DB 540 @ W / 010801

Vos références pour ce dossier : *(facultatif)*		706020c3SLC/AMD
6 MANDATAIRE *(s'il y a lieu)*		
Nom		LE CACHEUX
Prénom		Samuel
Cabinet ou Société		Cabinet BEAU de LOMENIE
N °de pouvoir permanent et/ou de lien contractuel		
Adresse	Rue	51, avenue Jean-Jaurès B. P. 7073
	Code postal et ville	6 9 3 0 1 LYON CEDEX 07
	Pays	France
N° de téléphone *(facultatif)*		0 472 76 85 30
N° de télécopie *(facultatif)*		04 78 69 86 82
Adresse électronique *(facultatif)*		contact@cabinetbeaudelomenie.fr
7 INVENTEUR (S)		**Les inventeurs sont nécessairement des personnes physiques**
Les demandeurs et les inventeurs sont les mêmes personnes		☐ Oui ☒ Non : **Dans ce cas remplir le formulaire de Désignation d'inventeur(s)**
8 RAPPORT DE RECHERCHE		**Uniquement pour une demande de brevet (y compris division et transformation)**
Établissement immédiat ou établissement différé		☐ ☒
Paiement échelonné de la redevance *(en deux versements)*		**Uniquement pour les personnes physiques effectuant elles-mêmes leur propre dépôt** ☐ Oui ☐ Non
9 RÉDUCTION DU TAUX DES REDEVANCES		**Uniquement pour les personnes physiques** ☐ Requise pour la première fois pour cette invention *(joindre un avis de non-imposition)* ☐ Obtenue antérieurement à ce dépôt pour cette invention *(joindre une copie de la décision d'admission à l'assistance gratuite ou indiquer sa référence)* : **AG**
Si vous avez utilisé l'imprimé «Suite», indiquez le nombre de pages jointes		

10 SIGNATURE DU DEMANDEUR OU DU MANDATAIRE (Nom et qualité du signataire)

Le Mandataire :
Samuel LE CACHEUX
CPI n° 00-0405

VISA DE LA PRÉFECTURE OU DE L'INPI

S. TEYSSEIRE

La loi n°78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés s'applique aux réponses faites à ce formulaire. Elle garantit un droit d'accès et de rectification pour les données vous concernant auprès de l'INPI.

La présente invention concerne, d'une part, le domaine technique de la conception, assistée par ordinateur (CAO), de systèmes électroniques ~~digitals~~ digitaux intégrés, encore appelés « puces électroniques » et, d'autre part, le domaine technique des puces électroniques obtenues.

De manière générale, la conception de systèmes électroniques complexes, destinés à être intégrés sur une même puce électronique, fait intervenir une phase d'élaboration d'une description du système électronique intégré dans un langage, dit de haut niveau (HDL – High level Description Language), à un niveau, dit de transfert des registres (RTL – Register Transfert Level). Les langages les plus communément utilisés, pour réaliser une telle description HDL, sont les langages Verilog ou VHDL, sans qu'il faille considérer que ces langages soient les seuls permettant une description HDL au niveau RTL d'un système électronique intégré.

La description d'un système électronique intégré en langages HDL se matérialise le plus souvent sous la forme d'un système de fichiers électroniques ou base de données de description pouvant alors être constitué par un seul et même fichier texte établi en langage HDL ou, au contraire, comprendre plusieurs fichiers textes de description, certains des fichiers correspondant à la description particulière de modules ou de parties du système intégré, tandis que d'autres fichiers décrivent l'interaction et les relations entre les différents modules et les liens existant entre ces derniers.

Pour obtenir une description de la puce électronique qui pourraient être qualifiée de matérielle par rapport à la description en langage HDL qui pourrait être qualifiée de fonctionnelle ou comportementale, il est réalisé, à partir du système de fichiers de description HDL, une synthèse ou compilation au moyen d'un outil informatique, généralement baptisé compilateur de silicium, permettant d'obtenir une description matérielle au niveau des portes logiques, en fonction de la technologie retenue, description encore appelée ~~« net-liste »~~ « netlist » qui sera ensuite utilisée pour obtenir une représentation physique du système électronique intégré sous la forme de masques permettant la fabrication de la puce, conformément aux

différentes techniques connues, ces dernières n'entrant pas dans le cadre de la présente invention.

Un système électronique digital intégré ainsi obtenu doit, bien entendu, offrir une garantie de fiabilité et de fonctionnement conforme à l'objectif visé lors de sa conception.

Ainsi, il apparaît nécessaire, lors de la conception d'un système électronique, de prévoir des systèmes ou des moyens permettant d'en vérifier le parfait fonctionnement, de manière bien entendue automatisée, soit au moyen de dispositifs extérieurs qui seront connectés au système électronique intégré, une fois ce dernier fabriqué, soit au moyen, de systèmes de tests faisant partie intégrante du système électronique intégré obtenu.

De manière générale, une telle démarche, orientée vers la testabilité des systèmes électroniques intégrés, est qualifiée de technique de DFT, pour « Design For Test » : conception pour le test, et, de manière plus particulière, lorsqu'il est prévu d'incorporer au système électronique intégré ses propres moyens de test automatique, on parle de BIST, pour « Built In Self Test » : auto test intégré.

Une première démarche, en vue de vérifier le bon fonctionnement d'un système électronique digital intégré, consiste, tout d'abord, à vérifier le parfait fonctionnement des éléments mémoire, bascules ou « flip-flop » présents au sein du système intégré et destinés à stocker, temporairement, des résultats intermédiaires de traitement ou des valeurs de signaux. Il s'agit ici d'éléments mémoire locaux présents au sein des composants dits séquentiels. Ces derniers représentent la majorité des circuits intégrés complexes tels que les microprocesseurs ou les processeurs de traitement de signal. Un circuit séquentiel étant composé d'éléments de logique combinatoire et d'éléments séquentiels ou bascules à distinguer des éléments mémoires des modules de mémoire vive RAM ou morte ROM.

Le test des circuits séquentiels passe par une étape de génération de vecteurs de tests en utilisant des outils logiciels spécialisés dits ATPG pour

« Automatic Test Pattern Generators ». La qualité des vecteurs de test générés détermine la phase de test après fabrication et la capacité des vecteurs de test à révéler la présence des défauts. La génération de vecteurs de test de qualité nécessite la prise en compte des techniques de DFT telle que le SCAN. La technique de « SCAN », consistant à chaîner entre eux les différents éléments mémoire, de manière à obtenir une ou plusieurs chaînes qui seront activées dans le cadre d'un fonctionnement en mode test du circuit intégré. -

La mise en place des fonctionnalités de SCAN et du chaînage des éléments mémoire peut intervenir au niveau de la description matérielle (netlist) du circuit électronique digital intégré comme décrit dans le brevet US 6,311,317. Toutefois, compte tenu du nombre très important de portes logiques notamment, cette insertion effectuée de manière automatique ou semi-automatique requiert un temps très important de calcul. De plus, cette insertion est susceptible de perturber le fonctionnement en mode normal du système logique électronique intégré, de sorte que, après avoir procédé à ce chaînage des éléments au niveau de la description matérielle netlist, il peut apparaître nécessaire de modifier la conception du circuit et donc de réécrire la description en langage HDL de ce dernier, pour ensuite procéder à une nouvelle compilation silicium et une nouvelle insertion du chaînage des éléments mémoire au niveau netlist.

Or, ce processus itératif, qui peut s'avérer très long et consommateur de ressources matérielles et humaines, constitue un obstacle à la réduction du temps nécessaire pour la conception de systèmes électroniques intégrés fiables et performants.

Ainsi, il est apparu que, si l'intégration des fonctionnalités de ~~scan~~ SCAN pouvaient être effectuées au niveau de la description HDL avant la phase de synthèse, il serait possible d'obtenir une réduction substantielle du temps de conception du circuit électronique intégré.

Ainsi, une autre voie a été proposée consistant à incorporer les fonctionnalités, dites de chaînage ou de ~~scan~~ SCAN, au niveau RTL, dans le cadre de la description HDL du système électronique digital intégré.

Le brevet US 6 256 770 a, par exemple, proposé un procédé et un dispositif de mise en œuvre de fonctionnalité de test d'un système électronique intégré dans le cadre de sa description en langage HDL, prévoyant tout d'abord d'attribuer des portions de chaînes d'éléments mémoire à différents modules du circuit, puis de procéder à un ordonnancement de ces portions de chaînes d'éléments mémoire sur la base d'une analyse des relations fonctionnelles existant entre les éléments mémoire ou les vecteurs de données dans les descriptions HDL des modules pour, enfin, sur la base de cet ordonnancement, procéder à une insertion des instructions de chaînage dans la description en langage HDL du module concerné, de manière que, lors de la synthèse dudit module, le système électronique digital intégré incorpore, pour chaque module concerné, les circuits électroniques logiques nécessaires au test qui découle d'un tel chaînage.

Un tel procédé et dispositif permet, effectivement, une insertion automatique d'instructions HDL permettant d'obtenir, lors de la synthèse du circuit, les fonctionnalités de SCAN, permettant d'assurer la génération de vecteurs de test de bonne qualité pour le circuit intégré sous test.

Toutefois, il est apparu à l'usage que l'étape d'analyse des relations fonctionnelles, existant entre les différents vecteurs de données, dans le cade de la conception de systèmes électroniques digitaux intégrés particulièrement complexes, induit un temps de calcul afin de procéder à cette analyse des relations fonctionnelles, particulièrement important, de sorte que les bénéfices de l'insertion au niveau RTL en langage HDL des fonctionnalités de SCAN se trouvent amoindris, voire annulés par les temps de calcul ou la puissance de calcul requise pour procéder à cette insertion, conformément au brevet US 6 256 770.

Une demande de brevet US 2003/0023941 présente une autre manière de procéder à l'insertion automatique au niveau RTL d'instructions en langage HDL, permettant de mettre en œuvre les fonctionnalités de ~~scan~~ SCAN dans le système électronique intégré qui sera obtenu par la synthèse de la description HDL ainsi modifiée.

Selon ce document, l'insertion des chaînes de ~~scan~~ SCAN et des points de test au niveau RTL en langage HDL est effectuée en réalisant, tout d'abord, une analyse de la testabilité de la description en langage HDL du système électronique intégré.

Or, si la méthode proposée par la demande US 2003/0023941 permet, effectivement, une insertion automatique des instructions HDL correspondant à des fonctionnalités de SCAN après synthèse, l'analyse de testabilité se trouve être une étape particulièrement consommatrice de ressources de calcul ou de temps, de sorte que les gains, obtenus par la modification automatique au niveau HDL du système électronique intégré, se trouvent dans ce cas également minimisés par les temps de calcul d'analyse de testabilité.

Par ailleurs, la demande US 2003/0023941 propose également de procéder à l'insertion des chaînes SCAN en effectuant une identification et une analyse des différents domaines d'horloge existants puis un calcul de minimisation des coûts de génération de test et de minimisation des domaines d'horloges. Or, cette analyse des domaines d'horloge et cette minimisation requiers également des ressources importantes.

Il apparaît donc le besoin d'une méthode qui, en assurant une insertion automatique dans le cadre de la description HDL au niveau RTL d'un système électronique digital intégré, des fonctionnalités de SCAN, permette de réduire substantiellement les temps de calcul, tout en offrant un système électronique digital intégré qui, après synthèse, présentera des performances au moins équivalentes à celles des systèmes intégrés qui seraient synthétisés à partir des descriptions HDL au niveau RTL traitées par les méthodes selon l'art antérieur.

Afin d'atteindre cet objectif, l'invention concerne un procédé d'analyse d'un ensemble de fichiers originaux de description d'un système électronique digital intégré dans un langage de description au niveau transfert de registres, dit langage HDL, en vue d'insérer de manière automatique dans les fichiers de description des instructions en langage HDL pour obtenir un nouvel ensemble de fichier de description en langage HDL du système électronique digital intégré incorporant des fonctionnalités de test de sorte que lors de la synthèse automatique du système électronique digital intégré à partir du nouvel ensemble de fichiers de description HDL le système électronique digital intégré obtenu incorpore une partie au moins les circuits électroniques logiques nécessaires au test du fonctionnement des éléments mémoires au moins.

Selon l'invention, le procédé d'analyse et d'insertion automatique est caractérisé en ce qu'il comprend les étapes suivantes:

- localisation automatique, dans les fichiers de description HDL orignaux des séquences d'instructions HDL qui, lors de la synthèse du système, seront à l'origine d'éléments mémoires,
- insertion, dans une partie au moins des fichiers de description HDL, de manière automatique séquentielle et sans analyse relationnelle ou fonctionnelle des éléments mémoire identifiés, d'instructions HDL assurant l'obtention, lors de la synthèse du système, d'une part, d'au moins une chaîne, dite de « SCAN », reliant les éléments mémoires et, d'autre part, des moyens de mise en œuvre du test dit de scan du circuit.

Au sens de l'invention, l'ensemble de fichiers de description HDL d'un système électronique digital intégré comprend un ou plusieurs fichiers de texte ou code ASCII qui décrivent en instruction HDL un, plusieurs ou tous les modules fonctionnels du système électronique digital intégré ainsi que les relations éventuelles existant entre les différents modules.

De même au sens de l'invention, il est effectué l'insertion de l'ensemble des instructions HDL nécessaire à la mise en œuvre du test dit de scan

SCAN à savoir notamment l'insertion des instructions permettant la mise du circuit à tester en mode de test, les instructions d'entrée de signal de test, de sortie de signal de test, les instructions de mise en œuvre d'une horloge de test, les instructions assurant le chaînage des éléments mémoire ainsi que les instructions de définition d'un contrôleur de test de SCAN sans que cette liste puisse être considérée comme possédant un caractère exhaustif ou exclusif d'autre fonctionnalité qui pourraient être nécessaire à la mise en œuvre du test.

Le procédé selon l'invention présente l'avantage du fait de l'insertion séquentielle des instructions de chaînage des éléments au fur et à mesure de leur apparition dans les pages de description HDL de ne pas nécessiter d'importantes ressources de calcul de sorte que le procédé selon l'invention peut être mis en œuvre sur un ordinateur, tel qu'un ordinateur personnel, tout en obtenant des temps de traitement moindres que ceux nécessaires pour la mise en œuvre des procédés selon l'art antérieur.

En effet, les inventeurs ont eu le mérite de mettre en évidence qu'il n'était pas nécessaire de procéder à une analyse, relationnelle ou fonctionnelle, ni même à une analyse de testabilité pour procéder à l'insertion des instructions HDL nécessaires à la mise en œuvre des fonctionnalités de SCAN et qu'une insertion séquentielle desdites instructions HDL, insertion qui pourrait être qualifiée d'insertion heuristique, au fur et à mesure de l'apparition dans les fichiers de descriptions HDL de ces instructions susceptibles d'engendrer des éléments de mémoire, permettait, en fin de compte, d'obtenir toutes les fonctionnalités de test des éléments mémoire du système électronique digital intégré sans en altérer les performances ni en augmenter de manière trop importante la surface.

Selon une caractéristique de l'invention, le procédé d'analyse et d'insertion automatique comprend une étape enregistrement du nouvel ensemble de fichiers de description HDL obtenus.

Selon une autre caractéristique de l'invention, afin d'éviter des violations des règles de ~~scan~~ SCAN lors de la synthèse du circuit à partir du

nouvel ensemble de fichiers de description HDL, le procédé d'analyse et d'insertion automatique comprend une phase d'identification des éventuels différents domaines d'horloge existants et l'étape d'insertion d'instructions HDL de chaînage d'éléments mémoire est alors réalisée de manière à créer au moins une chaîne de ~~scan~~ SCAN distincte pour chaque domaine d'horloge.

Par ailleurs, afin d'assurer une implémentation du ~~scan~~ SCAN au niveau RTL qui garantisse lors de la synthèse le respect des règles de ~~scan~~ SCAN, selon l'invention la dimension des variables ou signaux est déterminée avant l'étape d'insertion des instructions HDL de ~~scan~~ SCAN. Ainsi par exemple dans le cas de variable VHDL de type entier ou énumération, l'invention prévoit que la longueur des mots correspondant ou nombre de bits doit être fixé avant l'insertion des instructions VHDL de ~~scan~~ SCAN afin de garantir que les mémoires élémentaires constitutives de chaque mémoire sont bien chaînés entre-eux.

Ainsi, selon une autre caractéristique de l'invention, le procédé d'analyse et d'insertion automatique :

- comprend une étape d'analyse de l'ensemble de fichiers originaux de description HDL et de création d'au moins un fichier d'indexation comprenant, pour chaque objet et processus HDL, au moins le type et les coordonnées dans les fichiers de description HDL originaux,
- et l'étape de localisation des instructions HDL qui lors de la synthèse du circuit seront à l'origine d'éléments mémoires, comprend une phase de création d'un fichier de localisation des mémoires comprenant, pour chaque élément mémoire, au moins le nom de l'objet HDL correspondant, son type, sa dimension et ses coordonnées dans les fichiers de description HDL originaux.

De plus, dans la mesure où des informations sur la dimension de certaines variables seraient absentes de l'ensemble de fichiers originaux de description HDL, l'invention prévoit dans une forme préférée de mise en œuvre, une étape soit de définition automatique de cette dimension sur la

base d'une valeur par défaut prédéterminée soit de définition interactive avec un utilisateur du procédé.

Dans le même sens, et selon une variante de mise de œuvre préférée, le procédé conforme à l'invention vérifie, lors de l'insertion des instructions HDL de chaînage, la compatibilité des éléments mémoires entre eux. En effet, il n'est possible de chaîner que des éléments mémoires correspondant à des objets de même type et de ~~même~~ dimension compatible. Ainsi, en cas d'incompatibilité, l'invention prévoit de manière préférée mais non que l'étape d'insertion des instructions HDL de chaînage comprend soit une phase de transformation automatique du type et/ou de la dimension d'un ou des deux objets à l'origine du conflit, soit une phase de modification interactive avec l'utilisateur du type et/ou de la dimension d'un ou des deux objets à l'origine du conflit. En ce qui concerne la détection automatique et la correction de tels conflits correspondant à des erreurs de syntaxe ou grammaticales dans la mise en œuvre du langage, il est possible de se reporter à la demande de brevet US 2003/0033595.

Selon une autre caractéristique de l'invention, l'étape d'insertion d'instruction HDL de chaînage d'éléments mémoire comprend :

- une phase d'insertion d'instructions HDL de chaînage dit local d'éléments mémoires au niveau d'ensemble d'instructions HDL correspondant à un processus HDL de manière à obtenir lors de la synthèse au moins une chaîne distincte d'éléments mémoires pour chaque processus HDL,
- une phase d'insertion d'instruction HDL de chaînage, dit global, au niveau des fichiers de description HDL, de manière à obtenir, lors de la synthèse, au moins une chaîne d'éléments mémoire comprenant les chaînes d'éléments mémoire crées lors de la phase de chaînage local.

De manière général dans le cas du chaînage au sein même des processus on parle de chaînage dans le domaine séquentiel tandis que pour

le chaînage hors processus on parle de chaînage dans le domaine concurrent.

Ainsi, selon encore une caractéristique de l'invention, l'étape d'insertion automatique des instructions HDL comprend les phases suivantes :

- insertion d'instructions HDL correspondant à des signaux de test utilisés comme port d'entrée-sortie,
- insertion d'instructions HDL correspondant à des signaux intermédiaires de travail,
- insertion, au niveau de chaque processus, d'instructions HDL assurant l'obtention, lors de la synthèse du circuit, d'au moins une chaîne, dite de « SCAN», reliant les éléments mémoires propres au processus,
- insertion d'instructions HDL assurant une affectation concurrente des chaîne des entrées et sorties des chaîne de SCAN en dehors des processus.

Selon l'invention, le procédé d'analyse et d'insertion d'instructions HDL peut être mis en œuvre dans le cadre de différents langages de description HDL, tels que Verilog ou VHDL, étant entendu qu'il ne s'agit là que d'exemples non limitatifs et que le procédé selon l'invention pourrait être mis en œuvre pour encore d'autres langages de description HDL.

De plus, le procédé peut également être mis en œuvre sur un ensemble hétérogène de fichiers originaux de description en langage HDL comprenant par exemple mais non exclusivement des fichiers de description établis en langage Verilog et d'autres établis en langage VHDL.

Ainsi, selon une autre caractéristique de l'invention, dans le cas de l'utilisation des langages Verilog et VHDL en tant que langages de description HDL, l'étape de localisation des instructions HDL à l'origine des éléments mémoire comprend :

- une étape de recherche de processus synchronisés afin de détecter les objets affectés à l'intérieur de ces processus

- tout objet affecté à l'intérieur d'un processus et qui est lu dans un autre processus ou dans la partie concurrente du code HDL sera considéré comme un élément mémoire
- dans un processus synchronisé, tout objet affecté dans une branche d'une structure de contrôle « if » sans qu'il soit affecté dans toutes autres branches de cette même structure est considéré comme un élément mémoire
- dans un processus synchronisé, tout objet qui est lu avant d'être écrit est référencé comme un élément mémoire.

Dans une forme de mise en œuvre préférée du procédé selon l'invention et dans le cadre du chaînage local d'un processus décrit en langage VHDL, il est prévu une phase d'insertion d'instructions VHDL de définition de signaux intermédiaires destinés à reprendre les valeurs des chaînes de variables afin de permettre leur affectation et leur chaînage en dehors des processus.

Par ailleurs, selon l'invention, l'insertion automatique des instructions HDL doit être réalisée de manière à n'induire aucune dégradation fonctionnelle du code en langage HDL du système électronique digital intégré original.

Selon une autre caractéristique de l'invention, afin de permettre une optimisation des chaînes de SCAN et une amélioration de la couverture de fautes après synthèse du système électronique digital intégré à partir du nouvel ensemble de fichiers de description HDL, sans qu'il soit nécessaire de modifier à nouveau la description en langage HDL et de mettre à nouveau en œuvre le procédé selon l'invention et éviter ainsi un allongement du temps de conception du circuit, il est créé des chaînes de SCAN programmable. A cet effet, l'étape d'insertion des instructions HDL de SCAN comprend une phase d'insertion d'instructions HDL qui lors de la synthèse généreront un mutliplexeur programmable intercalé entre au moins les éléments mémoire d'une chaîne de SCAN. De manière préférée, il est intercalé un tel multiplexeur entre tous les éléments mémoires successifs des chaînes de

SCAN. Bien entendu, il est également procédé à l'insertion des instructions HDL correspondant à un contrôleur des multiplexeurs intercalés dans les chaînes de SCAN.

L'invention concerne également un système électronique digital intégré ou système monopuce qui comprend au moins un module fonctionnel de logique combinatoire et des éléments mémoires associés ainsi que des moyens de test de type SCAN comprenant au moins une chaîne d'éléments mémoires. Selon l'invention, le système électronique digital intégré est caractérisé en ce qu'il comprend des moyens de reconfiguration programmable de la chaîne SCAN.

Selon une autre caractéristique de l'invention, toujours en vue l'améliorer les capacités de test du circuit qui sera obtenu à partir du nouvel ensemble de fichier de description HDL, le procédé comprend une étape d'insertion d'instructions HDL dont la synthèse sera à l'origine de moyens intégrés d'auto test (BIST) du système électronique digital intégré. De tel moyens comprennent au moins un générateur automatique de vecteurs de test (TPG – Test Pattern Generator), des moyens d'analyse de la réponse du système électronique et des moyens de contrôle du test. Selon une caractéristique préférée de mise en œuvre de l'invention, le générateur automatique de vecteurs de test est conçu de manière que la séquence d'initialisation du registre à décalage à contre réaction linéaire plus communément connu par le PRPG (Parallel Random Pattern Generator) soit programmable. Par ailleurs, selon une caractéristique préférée de mise en oeuvre de l'invention, la structure de génération de vecteurs de test et celle de l'analyse des réponses se basent sur la structure de SCAN programmable citée ci-dessus.

L'invention concerne, également, un dispositif de conception automatisé en langage de description au niveau transfert de registres, dit langage HDL d'un système complet ou d'une partie de système électronique digital intégré, dispositif comprenant au moins une unité de calcul, une unité de mémoire et une unité de stockage de fichiers, caractérisé en ce que l'unité

de stockage comprend des fichiers de description en langage HDL du système ou de la partie de système électronique intégré et en ce que les unités de calcul et de mémoire sont adaptées pour générer, en mettant en œuvre le procédé selon l'invention et à partir des fichiers de description HDL, de nouveaux fichiers de description HDL du système ou de la partie de système qui incorporent des instructions HDL, de manière que le système ou la partie de système électronique digital intégré obtenu à partir des nouveaux fichiers incorpore une partie au moins des circuits électroniques logiques nécessaires au test du fonctionnement des éléments mémoire au moins.

Dans une forme préférée de réalisation le dispositif comprend un ordinateur personnel mettant en oeuvre un programme dont l'exécution permet la mise en œuvre du procédé selon l'invention.

L'invention concerne aussi un support de données lisibles par ordinateur sur lequel est enregistré un programme dont l'exécution par un ordinateur permet la mise en œuvre du procédé selon l'invention.

Les figures 1 et 2 illustrent des exemples d'organigramme de mise en œuvre du procédé selon l'invention.

Les figures 3 à 10 illustre des fichiers de description en langage HDL utilisés et générés au cours de la mise en œuvre du procédé selon l'invention.

1- Approche « Scan reconfigurable »

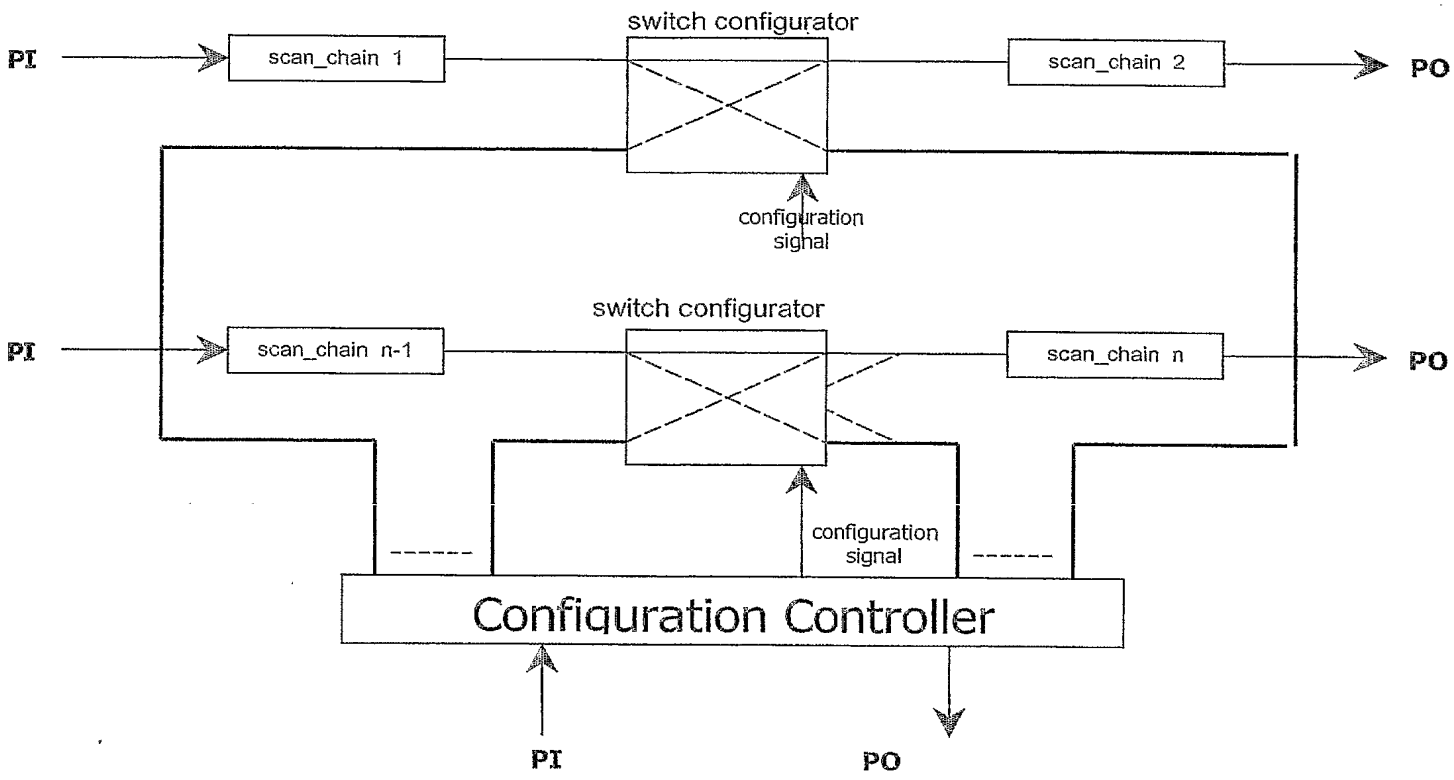
L'approche de scan reconfigurable consiste à permettre à un concepteur de revenir sur le choix lié à la configuration des chaînes de scan qui sont construites au niveau RTL. Ce choix peut être modifié soit au niveau logique une fois la « netlist » obtenue soit lors du test réel des circuits intégrés en post-production.

La reconfiguration des chaînes de scan consiste à redéfinir les paramètres suivants :

- Taille des chaînes de scan

- Configuration physique d'une ou de plusieurs chaîne de scan (éléments mémoires monolithiques constituant la chaîne de scan appelés « scan-chain » dans la figure).

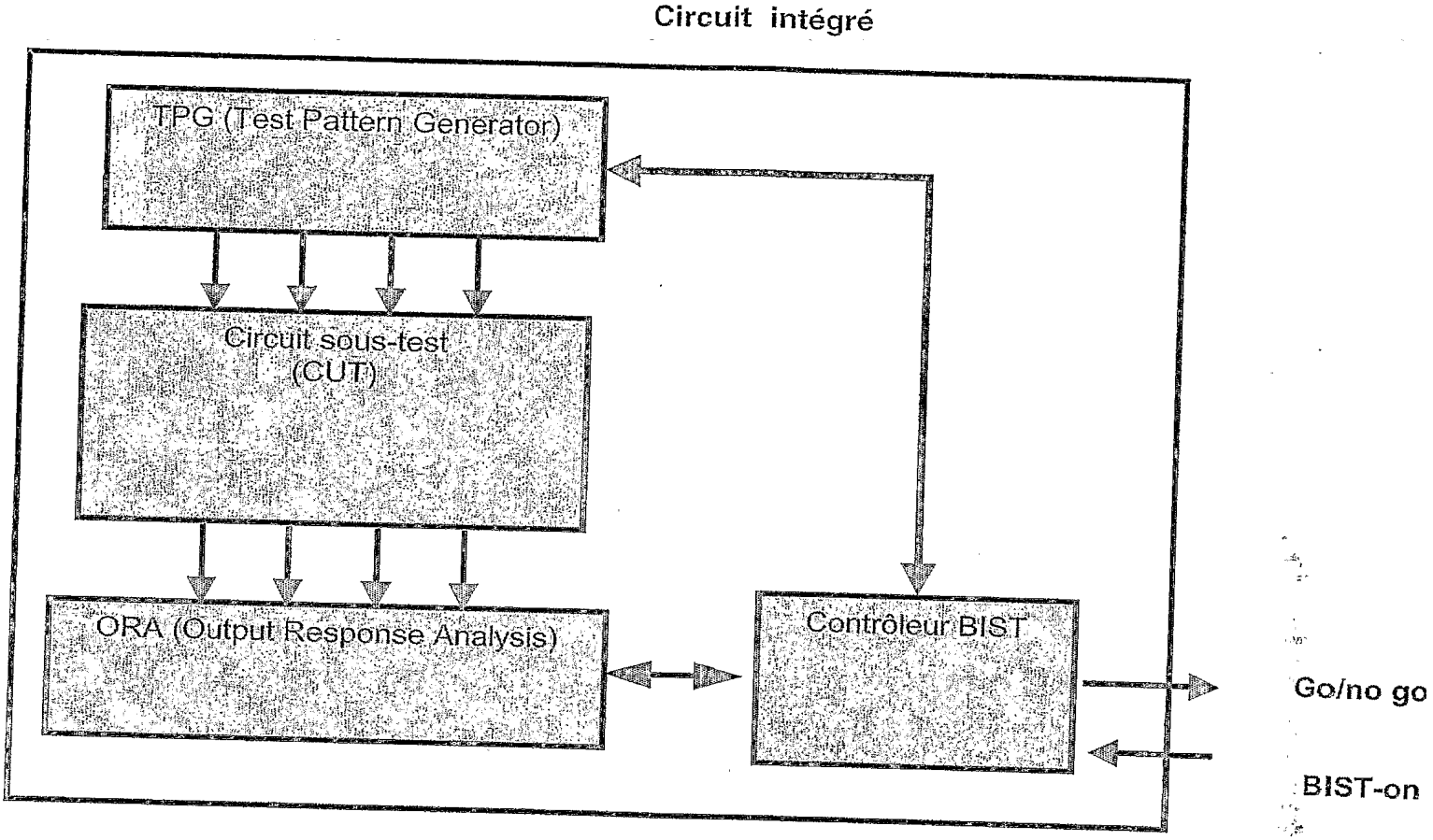
Comme illustré dans la figure ci-dessous, une telle reconfiguration passe par l'implantation au niveau RTL d'un contrôleur (configuration contrôleur) agissant sur les éléments de reconfiguration appelés commutateurs ou « switch configurators ». Le rôle d'un commutateur est de permettre l'activation d'une connexion selon la séquence de configuration activée au niveau du contrôleur.



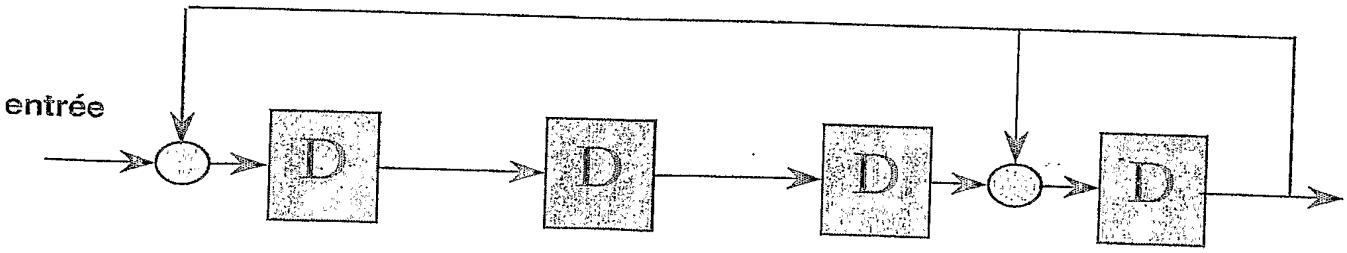
2- Approche « BIST et reprogrammation des séquences d'initialisation

15

L'architecture BIST est résumée dans la figure ci-dessous :



La structure de génération de vecteurs de test ou TPG ainsi que celle liée à la compression des résultats ou ORA est basée sur une structure dite de LFSR (Linear Feedback Shift Register) :



Les modules en couleur correspondent à des portes « ou exclusif » et celles en vert à des éléments de mémorisation ou bascules.

Contrairement à la structure ORA, aucune entrée extérieure n'est requise pour le TPG.

Concernant la structure de génération de vecteurs de test (TPG), la valeur initiale des bascules détermine la séquence de vecteurs de test générée et par conséquent la capacité de la séquence de test à détecter des défauts.
L'approche BIST d consiste à reprogrammer cette séquence d'initialisation une fois l'étape de synthèse logique effectuée. Les conditions de reprogrammation dépendent de la « netlist » obtenue ainsi que des résultats obtenus de l'outil de génération de vecteurs de test (ATPG). La reprogrammation de la séquence initiale passe par l'implantation au niveau RTL d'un contrôleur BIST qui agira directement sur les différences bascules constituant le LFSR.

REVENDICATIONS

1 - Procédé d'analyse d'un ensemble de fichiers originaux de description d'un système électronique digital intégré dans un langage de description au niveau transfert de registres, dit langage HDL, en vue d'insérer de manière automatique dans les fichiers de description des instructions en langage HDL pour obtenir un nouvel ensemble de fichier de description en langage HDL du système électronique digital intégré incorporant des fonctionnalités de test de sorte que lors de la synthèse automatique du système électronique digital intégré à partir du nouvel ensemble de fichiers le système électronique digital intégré obtenu incorpore une partie au moins les circuits électroniques logiques nécessaires au test du fonctionnement du circuit global,

procédé caractérisé en ce qu'il comprend les étapes suivantes :

- localisation automatique, dans les fichiers de description HDL orignaux des séquences d'instructions HDL qui, lors de la synthèse du système, seront à l'origine d'éléments mémoires,
- insertion, dans une partie au moins des fichiers de description HDL, de manière séquentielle automatique et sans analyse relationnelle ou fonctionnelle des éléments mémoire identifiés, d'instructions HDL assurant l'obtention, lors de la synthèse du système, d'au moins une chaîne, dite de « SCAN», reliant les éléments mémoires,
- enregistrement du nouvel ensemble de fichiers de description HDL obtenus.

2 - Procédé d'analyse et d'insertion automatique selon la revendication 1 ou 2, caractérisé en ce que l'étape d'insertion d'instruction HDL de chaînage d'éléments mémoires comprend :

- une phase d'insertion d'instructions HDL de chaînage dit local d'éléments mémoires au niveau d'ensemble d'instructions HDL correspondant à un objet HDL de manière à obtenir lors de la synthèse au moins une chaîne distincte d'éléments mémoires pour chaque objet HDL,

- une phase d'insertion d'instruction HDL de chaînage, dit global, au niveau des fichiers de description HDL, de manière à obtenir, lors de la synthèse, au moins une chaîne d'éléments mémoire comprenant les chaînes d'éléments mémoire crées lors de la phase de chaînage local.

3 - Procédé d'analyse et d'insertion automatique selon revendication 1 ou 2, caractérisé :

- en ce qu'il comprend une étape d'analyse de l'ensemble de fichiers originaux de description HDL et de création d'au moins un fichier d'indexation comprenant, pour chaque objet et processus HDL, la liste des unités de conception si elles existent (entité, librairie, paquetage), pour chaque unité de conception l'ensemble des déclarations, chaque déclaration comprenant le numéro de ligne, le nom de l'objet, son type, sa taille ainsi que le type de construction de contrôle associée,
- et en ce que l'étape de localisation des instructions HDL qui lors de la synthèse du circuit seront à l'origine d'éléments mémoires, comprend une phase de création d'un fichier de localisation des mémoires comprenant, pour chaque élément mémoire : le nom de l'objet, le fichier de référence, le type, la taille de l'objet ainsi que le nom de l'architecture.

4 - Procédé d'analyse et d'insertion automatique selon la revendication 3, caractérisé en que l'étape d'insertion automatique des instructions HDL comprend les phases suivantes :

- insertion d'instructions HDL correspondant à des signaux de test utilisés comme port d'entrée-sortie, lors du chaînage global,
- insertion d'instructions HDL correspondant à des signaux intermédiaires de travail ceci concerne le cas où il s'agit de chaîner des éléments de mémoire entre plusieurs processus où impliquant des ports primaires d'entrée/sortie.

- insertion, au niveau de chaque processus, d'instructions HDL assurant l'obtention, lors de la synthèse du circuit, d'au moins une chaîne, dite de «SCAN», reliant les éléments mémoires propres au processus,
- insertion d'instructions HDL assurant une affectation concurrente des pour le chaînage global des éléments mémoires en dehors des processus.

5 - Dispositif de conception automatisée en langage de description au niveau transfert de registres, dit langage HDL d'un système complet ou d'une partie de système électronique digital intégré, dispositif comprenant au moins une unité de calcul, une unité de mémoire et une unité de stockage de fichiers, caractérisé en ce que l'unité de stockage comprend des fichiers de description en langage HDL du système ou de la partie de système électronique intégré et en ce que les unités de calcul et de mémoire sont adaptées pour générer, en mettant en œuvre le procédé selon l'une des revendications à 1 à 4 et à partir des fichiers de description HDL, de nouveaux fichiers de description HDL du système ou de la partie de système qui incorporent des instructions HDL, de manière que le système ou la partie de système électronique digital intégré obtenu à partir des nouveaux fichiers incorpore une partie au moins des circuits électroniques logiques nécessaires au test du fonctionnement des éléments mémoire au moins.

Program modules:

Analyze – Parses the hdl source file and gets the necessary circuit structure information
Input: HDL source file; Output; VIF File, Project file
Get_memory – Detects the memory elements of the circuit
Input: VIF File; Output; MEM File
Build_Scan – Builds the scan chains according to the chosen method (Full Scan, Partial Scan, Custom Scan)
Input: MEM file; Output; Scan file or Reconstruction File
Build hierarchy model – Builds the circuit scanned hierarchical model
Input: Project File; Output; Reconstruction File
Reconstruction – Inserts the scan chains into the HDL source file and connects the hierarchical chains
Input: HDL source file, Reconstruction File; Output; Scanned HDL File

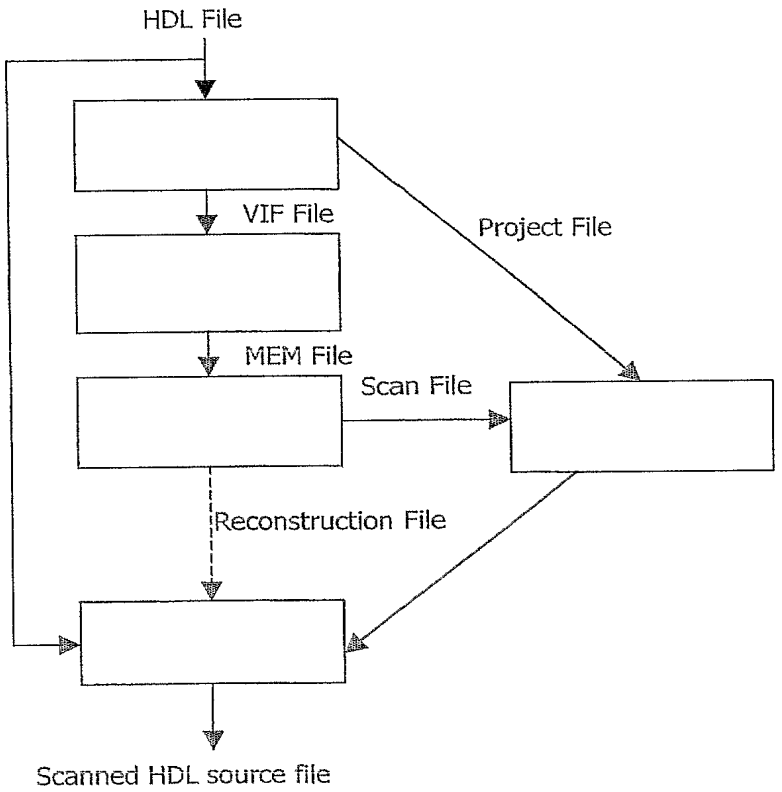


Figure 1 : organigramme procédé d'insertion automatique instruction de SCAN

Program modules:

System Graph generator – Given a VHDL or a Veilog description, it generates a graph where edges are related to combinational logic and nodes represent memory elements.

Testability Enhancement Block – This module adds new edges to the graph to enhance the controllability and the observability to nodes.

HiDFT-Scan – This is the software that generates scan automatically.

Pseudorandom test generation – it allows the selection of the size of test vectors and the parameters that are necessary for the test generation block

ATPG and compression architecture block – this block implemented in RTL allows the effective generation of test patterns and the compression of the obtained results. Final signatures are sent to the outputs through the scan chain.

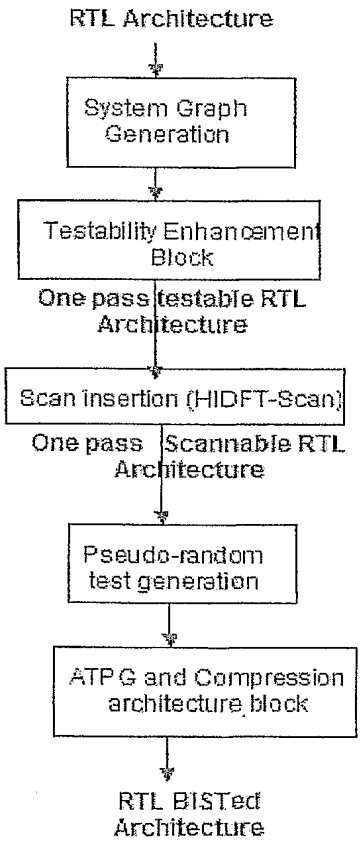


Figure 2 : Organigrame procédé

EXEMPLE fichier VHDL sans SCAN – page 1/2

```
\\exemple VHDL sans SCAN

library ieee;
use ieee.std_logic_1164.all;

entity b03 is

  port (
        CLOCK     : in std_logic;
        RESET     : in std_logic;
        request1  : in std_logic;
        request2  : in std_logic;
        request3  : in std_logic;
        request4  : in std_logic;
        grant_o   : out std_logic_vector(3 downto 0)
      );

end b03;

architecture BEHAV of b03 is

        constant INIT         : std_logic_vector(1 downto 0) :="00";
        constant ANALISI_REQ  : std_logic_vector(1 downto 0)  :="01";
        constant ASSIGN_CONST       : std_logic_vector(1 downto 0) :="10";
        signal c3                   : std_logic_vector(2 downto 0);

        constant U1         : std_logic_vector(2 downto 0) :="100";
        constant U2         : std_logic_vector(2 downto 0) :="010";
        constant U3         : std_logic_vector(2 downto 0) :="001";
        constant U4         : std_logic_vector(2 downto 0) :="111";

begin

process(CLOCK,RESET)


    variable coda0            : std_logic_vector(2 downto 0);
    variable coda1            : std_logic_vector(2 downto 0);
    variable coda2            : std_logic_vector(2 downto 0);
    variable coda3            : std_logic_vector(2 downto 0);
    variable stato            : std_logic_vector(1 downto 0);
    variable ru1,ru2,ru3,ru4  : std_logic;
    variable fu1,fu2,fu3,fu4  : std_logic;
    variable grant            : std_logic_vector(3 downto 0);

        begin
          if RESET='1' then
                        stato:=INIT;
                        coda0:="000";
                        coda1:="000";
                        coda2:="000";
                        coda3:="000";
                        ru1:='0';
                        fu1:='0';
                        ru2:='0';
                        fu2:='0';
                        ru3:='0';
                        fu3:='0';
                        ru4:='0';
                        fu4:='0';
                        grant:="0000";
                        grant_o<="0000";
          elsif CLOCK'event and CLOCK='1' then
          case stato is
                when ANALISI_REQ =>
                        c3<=coda3;
                        grant_o<=grant;

                 if (ru1='1')  then
                       if (fu1='0') then
                                coda3 := coda2;
                                coda2 := coda1;
                                coda1 := coda0;
                                coda0 := U1;
                       end if;
                 elsif (ru2='1')  then
                       if (fu2='0') then
                                coda3 := coda2;
                                coda2 := coda1;
                                coda1 := coda0;
```

EXEMPLE fichier VHDL sans SCAN – page 2/2

```
                                coda0 := U2;
                       end if;
                 elsif (ru3='1')  then
                       if (fu3='0') then
                                coda3 := coda2;
                                coda2 := coda1;
                                coda1 := coda0;
                                coda0 := U3;
                       end if;
                 elsif (ru4='1')  then
                       if (fu4='0') then
                                coda3 := coda2;
                                coda2 := coda1;
                                coda1 := coda0;
                                coda0 := U4;
                       end if;
                 end if;

                fu1:=ru1;
                fu2:=ru2;
                fu3:=ru3;
                fu4:=ru4;

                stato:=ASSIGN_CONST;

        when ASSIGN_CONST =>
                if ((fu1 or fu2 or fu3 or fu4)='1') then
                        case coda0 is
                                when U1 =>
                                        grant:="1000";
                                when U2 =>
                                        grant:="0100";
                                when U3 =>
                                        grant:="0010";
                                when U4 =>
                                        grant:="0001";
                                when others =>
                                        grant:="0000";
                        end case;
                        coda0:=coda1;
                        coda1:=coda2;
                        coda2:=coda3;
                        coda3:="000";
                end if;
                ru1  := request1;
                ru2  := request2;
                ru3  := request3;
                ru4  := request4;
                stato:= ANALISI_REQ;
        when INIT =>
                        ru1  := request1;
                        ru2  := request2;
                        ru3  := request3;
                        ru4  := request4;
                        stato:= ANALISI_REQ;
         when others =>

        end case;
        end if;
end process;

end BEHAV;
```

Fig 3

Exemple de fichier d'indexation VHDL – page 1/2

```
LIBRARY 1 { ieee } /type d'unité bibliothèque de ligne associée à l'unité /
USE 2 { ieee std_logic_1164 }
ENTITY 4 b03
DECLARATION 7 { CLOCK } INPUT std_logic (0:0) AFFECTED_BY { } /type-déclaration numéro-ligne nom-objet entrée-
sortie type taille trajet/
DECLARATION 8 { RESET } INPUT std_logic (0:0) AFFECTED_BY { }
DECLARATION 9 { request1 } INPUT std_logic (0:0) AFFECTED_BY { }
DECLARATION 10 { request2 } INPUT std_logic (0:0) AFFECTED_BY { }
DECLARATION 11 { request3 } INPUT std_logic (0:0) AFFECTED_BY { }
DECLARATION 12 { request4 } INPUT std_logic (0:0) AFFECTED_BY { }
DECLARATION 13 { grant_o } OUTPUT std_logic_vector (3:0) AFFECTED_BY { }
END ENTITY 16
ARCHITECTURE 18 BEHAV OF b03
DECLARATION 20 { INIT } CON std_logic_vector (1:0) AFFECTED_BY { }
DECLARATION 21 { ANALISI_REQ } CON std_logic_vector (1:0) AFFECTED_BY { }
DECLARATION 22 { ASSIGN_CONST } CON std_logic_vector (1:0) AFFECTED_BY { }
DECLARATION 23 { c3 } SIG std_logic_vector (2:0) AFFECTED_BY { }
DECLARATION 25 { U1 } CON std_logic_vector (2:0) AFFECTED_BY { }
DECLARATION 26 { U2 } CON std_logic_vector (2:0) AFFECTED_BY { }
DECLARATION 27 { U3 } CON std_logic_vector (2:0) AFFECTED_BY { }
DECLARATION 28 { U4 } CON std_logic_vector (2:0) AFFECTED_BY { }
PROCESS 35 { CLOCK RESET }
DECLARATION 35 { coda0 } VAR std_logic_vector (2:0) AFFECTED_BY { }
DECLARATION 36 { coda1 } VAR std_logic_vector (2:0) AFFECTED_BY { }
DECLARATION 37 { coda2 } VAR std_logic_vector (2:0) AFFECTED_BY { }
DECLARATION 38 { coda3 } VAR std_logic_vector (2:0) AFFECTED_BY { }
DECLARATION 39 { stato } VAR std_logic_vector (1:0) AFFECTED_BY { }
DECLARATION 40 { ru1 ru2 ru3 ru4 } VAR std_logic (0:0) AFFECTED_BY { }
DECLARATION 41 { fu1 fu2 fu3 fu4 } VAR std_logic (0:0) AFFECTED_BY { }
DECLARATION 42 { grant } VAR std_logic_vector (3:0) AFFECTED_BY { }
INSTRUCTION 45 IF { RESET }
INSTRUCTION 46 AFFECT { stato } AFFECTED_BY { INIT }
INSTRUCTION 47 AFFECT { coda0 } AFFECTED_BY { }
INSTRUCTION 48 AFFECT { coda1 } AFFECTED_BY { }
INSTRUCTION 49 AFFECT { coda2 } AFFECTED_BY { }
INSTRUCTION 50 AFFECT { coda3 } AFFECTED_BY { }
INSTRUCTION 51 AFFECT { ru1 } AFFECTED_BY { }
INSTRUCTION 52 AFFECT { fu1 } AFFECTED_BY { }
INSTRUCTION 53 AFFECT { ru2 } AFFECTED_BY { }
INSTRUCTION 54 AFFECT { fu2 } AFFECTED_BY { }
INSTRUCTION 55 AFFECT { ru3 } AFFECTED_BY { }
INSTRUCTION 56 AFFECT { fu3 } AFFECTED_BY { }
INSTRUCTION 57 AFFECT { ru4 } AFFECTED_BY { }
INSTRUCTION 58 AFFECT { fu4 } AFFECTED_BY { }
INSTRUCTION 59 AFFECT { grant } AFFECTED_BY { }
INSTRUCTION 60 ASSIGN { grant_o } ASSIGNED_BY { }
SINCRO_CLK 61 CLOCK
INSTRUCTION 61 ELSIF { CLOCK }
INSTRUCTION 62 CASE { stato }
INSTRUCTION 63 WHEN { ANALISI_REQ }
INSTRUCTION 64 ASSIGN { c3 } ASSIGNED_BY { coda3 }
INSTRUCTION 65 ASSIGN { grant_o } ASSIGNED_BY { grant }
INSTRUCTION 67 IF { ru1 }
INSTRUCTION 68 IF { fu1 }
INSTRUCTION 69 AFFECT { coda3 } AFFECTED_BY { coda2 }
INSTRUCTION 70 AFFECT { coda2 } AFFECTED_BY { coda1 }
INSTRUCTION 71 AFFECT { coda1 } AFFECTED_BY { coda0 }
INSTRUCTION 72 AFFECT { coda0 } AFFECTED_BY { U1 }
INSTRUCTION 74 ELSIF { ru2 }
INSTRUCTION 75 IF { fu2 }
INSTRUCTION 76 AFFECT { coda3 } AFFECTED_BY { coda2 }
INSTRUCTION 77 AFFECT { coda2 } AFFECTED_BY { coda1 }
INSTRUCTION 78 AFFECT { coda1 } AFFECTED_BY { coda0 }
INSTRUCTION 79 AFFECT { coda0 } AFFECTED_BY { U2 }
INSTRUCTION 81 ELSIF { ru3 }
INSTRUCTION 82 IF { fu3 }
INSTRUCTION 83 AFFECT { coda3 } AFFECTED_BY { coda2 }
INSTRUCTION 84 AFFECT { coda2 } AFFECTED_BY { coda1 }
INSTRUCTION 85 AFFECT { coda1 } AFFECTED_BY { coda0 }
INSTRUCTION 86 AFFECT { coda0 } AFFECTED_BY { U3 }
INSTRUCTION 88 ELSIF { ru4 }
INSTRUCTION 89 IF { fu4 }
INSTRUCTION 90 AFFECT { coda3 } AFFECTED_BY { coda2 }
INSTRUCTION 91 AFFECT { coda2 } AFFECTED_BY { coda1 }
INSTRUCTION 92 AFFECT { coda1 } AFFECTED_BY { coda0 }
INSTRUCTION 93 AFFECT { coda0 } AFFECTED_BY { U4 }
```

Fig 4

Exemple de fichier d'indexation VHDL – page 2/2

```
INSTRUCTION 97 AFFECT { fu1 } AFFECTED_BY { ru1 }
INSTRUCTION 98 AFFECT { fu2 } AFFECTED_BY { ru2 }
INSTRUCTION 99 AFFECT { fu3 } AFFECTED_BY { ru3 }
INSTRUCTION 100 AFFECT { fu4 } AFFECTED_BY { ru4 }
INSTRUCTION 102 AFFECT { stato } AFFECTED_BY { ASSIGN_CONST }
INSTRUCTION 104 WHEN { ASSIGN_CONST }
INSTRUCTION 105 IF { fu1 fu2 fu3 fu4 }
INSTRUCTION 106 CASE { coda0 }
INSTRUCTION 107 WHEN { U1 }
INSTRUCTION 108 AFFECT { grant } AFFECTED_BY { }
INSTRUCTION 109 WHEN { U2 }
INSTRUCTION 110 AFFECT { grant } AFFECTED_BY { }
INSTRUCTION 111 WHEN { U3 }
INSTRUCTION 112 AFFECT { grant } AFFECTED_BY { }
INSTRUCTION 113 WHEN { U4 }
INSTRUCTION 114 AFFECT { grant } AFFECTED_BY { }
INSTRUCTION 115 WHEN { }
INSTRUCTION 116 AFFECT { grant } AFFECTED_BY { }
END CASE 117
INSTRUCTION 118 AFFECT { coda0 } AFFECTED_BY { coda1 }
INSTRUCTION 119 AFFECT { coda1 } AFFECTED_BY { coda2 }
INSTRUCTION 120 AFFECT { coda2 } AFFECTED_BY { coda3 }
INSTRUCTION 121 AFFECT { coda3 } AFFECTED_BY { }
INSTRUCTION 123 AFFECT { ru1 } AFFECTED_BY { request1 }
INSTRUCTION 124 AFFECT { ru2 } AFFECTED_BY { request2 }
INSTRUCTION 125 AFFECT { ru3 } AFFECTED_BY { request3 }
INSTRUCTION 126 AFFECT { ru4 } AFFECTED_BY { request4 }
INSTRUCTION 127 AFFECT { stato } AFFECTED_BY { ANALISI_REQ }
INSTRUCTION 128 WHEN { INIT }
INSTRUCTION 129 AFFECT { ru1 } AFFECTED_BY { request1 }
INSTRUCTION 130 AFFECT { ru2 } AFFECTED_BY { request2 }
INSTRUCTION 131 AFFECT { ru3 } AFFECTED_BY { request3 }
INSTRUCTION 132 AFFECT { ru4 } AFFECTED_BY { request4 }
INSTRUCTION 133 AFFECT { stato } AFFECTED_BY { ANALISI_REQ }
INSTRUCTION 134 WHEN { }
END CASE 136
END PROCESS 138
END ARCHITECTURE 140
```

1er dépôt 5/10

exemple-mem-vhdl

```
S_O grant_o CLOCK 35 std_logic_vector (3:0) b03 BEHAV /signal-variable nom
'horloge synchronisation type taille ref entité nom architecture /
VAR coda0 CLOCK 35 std_logic_vector (2 0) b03 BEHAV
VAR coda1 CLOCK 35 std_logic_vector (2:0) b03 BEHAV
VAR coda2 CLOCK 35 std_logic_vector (2 0) b03 BEHAV
VAR coda3 CLOCK 35 std_logic_vector (2.0) b03 BEHAV
VAR fu1 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR fu2 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR fu3 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR fu4 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR grant CLOCK 35 std_logic_vector (3:0) b03 BEHAV
VAR ru1 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR ru2 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR ru3 CLOCK 35 std_logic (0 0) b03 BEHAV
VAR ru4 CLOCK 35 std_logic (0.0) b03 BEHAV
VAR stato CLOCK 35 std_logic_vector (1:0) b03 BEHAV
PROCESS 1
```

fig 5

EXEMPLE Fichier VHDL Avec SCAN page 1/4

```
library ieee;
use ieee.std_logic_1164.all;

entity b03 is

  port (

--Declaration of scan_en, scan_in, scan_out----------
-----------------------------
SCAN_EN : in std_logic;
SCAN_I_IN : in std_logic;
SCAN_I_OUT : out std_logic;
-----------------------------


        CLOCK     : in std_logic;
        RESET     : in std_logic;
        request1  : in std_logic;
        request2  : in std_logic;
        request3  : in std_logic;
        request4  : in std_logic;
        grant_o   : out std_logic_vector(3 downto 0)
      );

end b03;

architecture BEHAV of b03 is

--Internal scan signals declaration-------------
signal grant_o_scan : std_logic_vector (3 downto 0);
-----------------------------------------------


     constant INIT        : std_logic_vector(1 downto 0) :="00";
     constant ANALISI_REQ : std_logic_vector(1 downto 0) :="01";
     constant ASSIGN_CONST      : std_logic_vector(1 downto 0) :="10";
     signal c3            : std_logic_vector(2 downto 0);

     constant U1          : std_logic_vector(2 downto 0) :="100";
     constant U2          : std_logic_vector(2 downto 0) :="010";
     constant U3          : std_logic_vector(2 downto 0) :="001";
     constant U4          : std_logic_vector(2 downto 0) :="111";

begin

process(CLOCK,RESET)


   variable coda0            : std_logic_vector(2 downto 0);
   variable coda1            : std_logic_vector(2 downto 0);
   variable coda2            : std_logic_vector(2 downto 0);
   variable coda3            : std_logic_vector(2 downto 0);
   variable stato            : std_logic_vector(1 downto 0);
   variable ru1,ru2,ru3,ru4  : std_logic;
   variable fu1,fu2,fu3,fu4  : std_logic;
   variable grant            : std_logic_vector(3 downto 0);

     begin
```

A

EXEMPLE Fichier VHDL Avec SCAN page 2/4

```
      if RESET='1' then
                stato:=INIT;
                coda0:="000";
                coda1:="000";
                coda2:="000";
                coda3:="000";
                ru1:='0';
                fu1:='0';
                ru2:='0';
                fu2:='0';
                ru3:='0';
                fu3:='0';
                ru4:='0';
                fu4:='0';
                grant:="0000";
                grant_o_scan <="0000";
      elsif CLOCK'event and CLOCK='1' then
case SCAN_EN is
when '1' =>
--Scan mode----------

      grant_o_scan(3)<=grant_o_scan(2);
      grant_o_scan(2)<=grant_o_scan(1);
      grant_o_scan(1)<=grant_o_scan(0);
      grant_o_scan(0)<=coda0(2);
      coda0(2):=coda0(1);
      coda0(1):=coda0(0);
      coda0(0):=coda1(2);
      coda1(2):=coda1(1);
      coda1(1):=coda1(0);
      coda1(0):=coda2(2);
      coda2(2):=coda2(1);
      coda2(1):=coda2(0);
      coda2(0):=coda3(2);
      coda2(2):=coda3(1);
      coda2(1):=coda3(0);
      coda3(0):=fu1;
      fu1:=fu2;
      fu2:=fu3;
      fu3:=fu4;
      fu4:=grant(3);
      grant(3):=grant(2);
      grant(2):=grant(1);
      grant(1):=grant(0);
      grant(0):=ru1;
      ru1:=ru2;
      ru2:=ru3;
      ru3:=ru4;
      ru4:=stato(1);
      stato(1):=stato(0);
      stato(0):=SCAN_I_IN;

when others =>
--Normal mode----------

      case stato is
           when ANALISI_REQ =>
                c3 <=coda3;
                grant_o_scan <=grant;
```

C

EXEMPLE Fichier VHDL Avec SCAN page 3/4

```
                if (ru1='1')  then
                if (fu1='0') then
                     coda3 := coda2;
                     coda2 := coda1;
                     coda1 := coda0;
                     coda0 := U1;
                end if;
                  elsif (ru2='1')  then
                if (fu2='0') then
                     coda3 := coda2;
                     coda2 := coda1;
                     coda1 := coda0;
                     coda0 := U2;
                end if;
                  elsif (ru3='1')  then
                if (fu3='0') then
                     coda3 := coda2;
                     coda2 := coda1;
                     coda1 := coda0;
                     coda0 := U3;
                end if;
                  elsif (ru4='1')  then
                if (fu4='0') then
                     coda3 := coda2;
                     coda2 := coda1;
                     coda1 := coda0;
                     coda0 := U4;
                end if;
                  end if;

           fu1:=ru1;
           fu2:=ru2;
           fu3:=ru3;
           fu4:=ru4;

                stato:=ASSIGN_CONST;

      when ASSIGN_CONST =>
                if ((fu1 or fu2 or fu3 or fu4)='1') then
           case coda0 is
                when U1 =>
                     grant:="1000";
                when U2 =>
                     grant:="0100";
                when U3 =>
                     grant:="0010";
                when U4 =>
                     grant:="0001";
                when others =>
                     grant:="0000";
           end case;
           coda0:=coda1;
           coda1:=coda2;
           coda2:=coda3;
           coda3:="000";
      end if;
      ru1  := request1;
      ru2  := request2;
      ru3  := request3;
      ru4  := request4;
```

B

EXEMPLE Fichier VHDL Avec SCAN page 4/4

```
                stato:= ANALISI_REQ;
           when INIT =>
                          ru1   := request1;
                          ru2   := request2;
                          ru3   := request3;
                          ru4   := request4;
                          stato:= ANALISI_REQ;
                when others =>

      end case;
      end case;
--End mod scan----------
end  if;
end process;


--Concurential assignment of the internal scan signals ----------
grant_o <= grant_o_scan;
SCAN_I_OUT<=grant_o_scan(3);

-----------------------------------------------------------------

end BEHAV;
```

Fig 6

D

EXEMPLE de fichier multiprocess Verilog sans scan

```
// Example of Multiple Processes Verilog sans scan


module example_4_processes (RESET, CLOCK, ENABLE, D_IN,
                  A_Q_OUT, B_Q_OUT, C_Q_OUT, D_Q_OUT);

input RESET, CLOCK, ENABLE;
input       [7:0] D_IN;
output      [7:0] A_Q_OUT;
output      [7:0] B_Q_OUT;
output      [7:0] C_Q_OUT;
output      [7:0] D_Q_OUT;

reg         [7:0] A_Q_OUT;
reg         [7:0] B_Q_OUT;
reg         [7:0] C_Q_OUT;
reg         [7:0] D_Q_OUT;

    // D flip-flop
    always @(posedge CLOCK)
    begin
        A_Q_OUT = D_IN;
    end

    // Flip-flop with asynchronous reset
    always @(posedge CLOCK)
    begin
        if (RESET)
            B_Q_OUT = 8'b00000000;
            else
               B_Q_OUT = D_IN;
    end

    // Flip-flop with asynchronous set
    always @(posedge CLOCK)
    begin
        if (RESET)
            C_Q_OUT = 8'b11111111;
            else
                C_Q_OUT = D_IN;
    end

    //Flip-flop with asynchronous reset & clock enable
    always @(posedge CLOCK)
    begin
            if (RESET)
                 D_Q_OUT = 8'b00000000;
            else if (ENABLE)
                 D_Q_OUT = D_IN;
    end

endmodule



EOF
```

Fig 7

Exemple fichier indexation Verilog - page 1/1

```
MODULE 5 example_4_processes { A_Q_OUT B_Q_OUT CLOCK C_Q_OUT D_IN D_Q_OUT ENABLE
RESET }
DECLARATION 7 INPUT $0:05 { CLOCK ENABLE RESET }
DECLARATION 8 INPUT $7:05 { D_IN }
DECLARATION 9 OUTPUT $7:05 { A_Q_OUT }
DECLARATION 10 OUTPUT $7:05 { B_Q_OUT }
DECLARATION 11 OUTPUT $7:05 { C_Q_OUT }
DECLARATION 12 OUTPUT $7:05 { D_Q_OUT }
DECLARATION 14  REG $7:05 { A_Q_OUT }
DECLARATION 15  REG $7:05 { B_Q_OUT }
DECLARATION 16  REG $7:05 { C_Q_OUT }
DECLARATION 17  REG $7:05 { D_Q_OUT }
PROCESS 20 { CLOCK }
SINCRO_CLK 20 { CLOCK }
INSTRUCTION 22 AFFECT { A_Q_OUT } AFFECTED_BY { D_IN }
END PROCESS 23
PROCESS 26 { CLOCK }
SINCRO_CLK 26 { CLOCK }
BEGIN_SECV 28 CLOCK
INSTRUCTION 28 IF { RESET }
INSTRUCTION 29 AFFECT { B_Q_OUT } AFFECTED_BY { }
INSTRUCTION 30 ELSE
INSTRUCTION 31 AFFECT { B_Q_OUT } AFFECTED_BY { D_IN }
END PROCESS 32
PROCESS 35 { CLOCK }
SINCRO_CLK 35 { CLOCK }
BEGIN_SECV 37 CLOCK
INSTRUCTION 37 IF { RESET }
INSTRUCTION 38 AFFECT { C_Q_OUT } AFFECTED_BY { }
INSTRUCTION 39 ELSE
INSTRUCTION 40 AFFECT { C_Q_OUT } AFFECTED_BY { D_IN }
END PROCESS 41
PROCESS 44 { CLOCK }
SINCRO_CLK 44 { CLOCK }
BEGIN_SECV 46 CLOCK
INSTRUCTION 46 IF { RESET }
INSTRUCTION 47 AFFECT { D_Q_OUT } AFFECTED_BY { }
INSTRUCTION 48 ELSE
INSTRUCTION 48 IF { ENABLE }
INSTRUCTION 49 AFFECT { D_Q_OUT } AFFECTED_BY { D_IN }
END PROCESS 50
ENDMODULE 52 example_4_processes
```

Fig 8

Exemple de fichier de localisation des éléments mémoire Verilog

```
exemple-mem-verilog


S_0 A_Q_OUT CLOCK 20 REG (7:0) example_4_processes
S_0 B_Q_OUT CLOCK 26 REG (7:0) example_4_processes
S_0 C_Q_OUT CLOCK 35 REG (7:0) example_4_processes
S_0 D_Q_OUT CLOCK 44 REG (7:0) example_4_processes
PROCESS 4
EOF
```

Fig 9

EXEMPLE de fichier multiprocess Verilog – page 1/2

```
// Example of Multiple Processes Verilog

module example_4_processes ( SCAN_EN, SCAN_IN_0001, SCAN_OUT_0001,  SCAN_IN_0002, SCAN_OUT_0002,
SCAN_IN_0003, SCAN_OUT_0003,  SCAN_IN_0004, SCAN_OUT_0004, RESET, CLOCK, ENABLE, D_IN,
                A_Q_OUT, B_Q_OUT, C_Q_OUT, D_Q_OUT);

//Declaration of scan_en, scan_in, scan_out
//************************
input SCAN_EN;
input SCAN_IN_0001;
output SCAN_OUT_0001;
reg SCAN_OUT_0001;
input SCAN_IN_0002;
output SCAN_OUT_0002;
reg SCAN_OUT_0002;
input SCAN_IN_0003;
output SCAN_OUT_0003;
reg SCAN_OUT_0003;
input SCAN_IN_0004;
output SCAN_OUT_0004;
reg SCAN_OUT_0004;
//************************


input RESET, CLOCK, ENABLE;
input       [7:0] D_IN;
output      [7:0] A_Q_OUT;
output      [7:0] B_Q_OUT;
output      [7:0] C_Q_OUT;
output      [7:0] D_Q_OUT;

reg         [7:0] A_Q_OUT;
reg         [7:0] B_Q_OUT;
reg         [7:0] C_Q_OUT;
reg         [7:0] D_Q_OUT;

    // D flip-flop
    always @(posedge CLOCK)
begin
if(SCAN_EN) begin
        SCAN_OUT_0001=A_Q_OUT[7];
        A_Q_OUT[7]=A_Q_OUT[6];
        A_Q_OUT[6]=A_Q_OUT[5];
        A_Q_OUT[5]=A_Q_OUT[4];
        A_Q_OUT[4]=A_Q_OUT[3];
        A_Q_OUT[3]=A_Q_OUT[2];
        A_Q_OUT[2]=A_Q_OUT[1];
        A_Q_OUT[1]=A_Q_OUT[0];
        A_Q_OUT[0]=SCAN_IN_0001;

end
else
    begin
        A_Q_OUT = D_IN;
    end
end
//end scan condition

    // Flip-flop with asynchronous reset
    always @(posedge CLOCK)
begin
if(SCAN_EN) begin
        SCAN_OUT_0002=B_Q_OUT[7];
        B_Q_OUT[7]=B_Q_OUT[6];
        B_Q_OUT[6]=B_Q_OUT[5];
        B_Q_OUT[5]=B_Q_OUT[4];
        B_Q_OUT[4]=B_Q_OUT[3];
        B_Q_OUT[3]=B_Q_OUT[2];
        B_Q_OUT[2]=B_Q_OUT[1];
        B_Q_OUT[1]=B_Q_OUT[0];
        B_Q_OUT[0]=SCAN_IN_0002;

end
else
    begin
        if (RESET)
            B_Q_OUT = 8'b00000000;
```

EXEMPLE de fichier multiprocess Verilog – page 2/2

```
            else
                B_Q_OUT = D_IN;
    end
end
//end scan condition

    // Flip-flop with asynchronous set
    always @(posedge CLOCK)
begin
if(SCAN_EN) begin
        SCAN_OUT_0003=C_Q_OUT[7];
        C_Q_OUT[7]=C_Q_OUT[6];
        C_Q_OUT[6]=C_Q_OUT[5];
        C_Q_OUT[5]=C_Q_OUT[4];
        C_Q_OUT[4]=C_Q_OUT[3];
        C_Q_OUT[3]=C_Q_OUT[2];
        C_Q_OUT[2]=C_Q_OUT[1];
        C_Q_OUT[1]=C_Q_OUT[0];
        C_Q_OUT[0]=SCAN_IN_0003;

end
else
    begin
        if (RESET)
            C_Q_OUT = 8'b11111111;
            else
                C_Q_OUT = D_IN;
    end
end
//end scan condition

    //Flip-flop with asynchronous reset & clock enable
    always @(posedge CLOCK)
begin
if(SCAN_EN) begin
        SCAN_OUT_0004=D_Q_OUT[7];
        D_Q_OUT[7]=D_Q_OUT[6];
        D_Q_OUT[6]=D_Q_OUT[5];
        D_Q_OUT[5]=D_Q_OUT[4];
        D_Q_OUT[4]=D_Q_OUT[3];
        D_Q_OUT[3]=D_Q_OUT[2];
        D_Q_OUT[2]=D_Q_OUT[1];
        D_Q_OUT[1]=D_Q_OUT[0];
        D_Q_OUT[0]=SCAN_IN_0004;

end
else
    begin
            if (RESET)
                D_Q_OUT = 8'b00000000;
            else if (ENABLE)
                D_Q_OUT = D_IN;
    end
end
//end scan condition

endmodule
```

Fig 10